

## Introducción

Una actividad es el componente encargado de la mayor parte de la interacción con el usuario en las aplicaciones Android. Con el fin de estudiar dicho elemento, en este tutorial veremos su concepto, el ciclo de vida que recorren en el sistema operativo y sus posibles estados. Así mismo iremos viendo un ejemplo en Android Studio que nos permita practicar su uso. Practicaremos también las opciones de multilinguaje de nuestra aplicación.

## Definición De Actividad

En cuestiones de diseño y navegación la actividad es el bloque de construcción encargado por la creación de una ventana que nuestra aplicación usa para dibujar y recibir eventos del sistema (gestos táctiles como tap, swipe, press and hold, etc.)

Android está hecho para que el usuario esté centrado en una sola actividad para que sea la única cosa que esté haciendo, lo que quiere decir que tan solo una puede estar en primer plano. Y Las demás se mueven a segunda instancia. Esta capacidad es originada por el **ciclo de vida** de las actividades que veremos más adelante.

Otro punto a tener en cuenta es que si tu app tiene varias actividades es posible marcar una como la **actividad principal** (main). Esto nos permite que al presionar el icono de la app desde el **Launcher**, dicha actividad sea la iniciada.

Debido a que la actividad es un bloque de construcción de una app, se convierte en un componente necesario a registrar en el archivo **AndroidManifest.xml**.

Para hacerlo incluimos dentro del nodo `<application>` la etiqueta `<activity>` como se muestra en el siguiente código:

```
<activity
    android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

La etiqueta de la actividad tiene una gran cantidad de atributos, pero el único que debe incluirse es `android:name`. Su valor es el nombre de clase, el cual puede ser escrito con el paquete completo (ej. `com.ejemplo.app1.MainActivity`), o usar el carácter punto para dejar que se complete con el paquete usado en la etiqueta `<manifest>` (ej. `.MainActivity`).

## Ciclo De Vida De Una Actividad

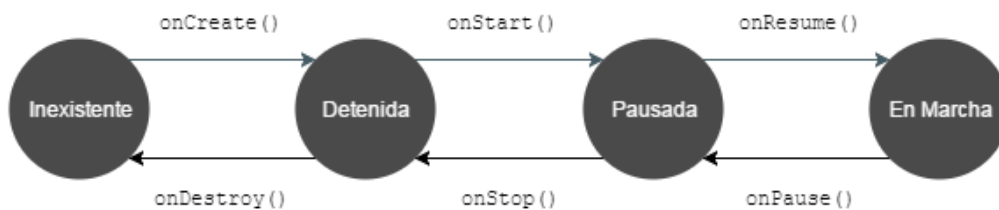
Debido al diseño del sistema operativo Android, una actividad puede atravesar los siguientes estados para mantener el flujo entre apps y eventos del sistema:

- Inexistente
- Detenida
- Pausada
- En marcha

La siguiente tabla nos muestra que sucede en memoria, UI y el proceso de primer plano, cuando la actividad pasa por su ciclo de vida:

Estado	¿En memoria?	¿Visible al usuario?	¿En primer plano?
Inexistente	No	No	No
Detenida	Si	No	No
Pausada	Si	Si	No
En Marcha	Si	Si	Si

El cambio entre un estado es disparado a través de métodos tipo callback llamados por Android). En este diagrama podemos ver los nombres de dichos métodos cuando una actividad va desde la construcción hasta su destrucción:



## Método onCreate()

Se dispara cuando el sistema crea una nueva instancia en memoria de la actividad.

Es el lugar donde se incluye la mayor de la inicialización. Tareas prioritarias como:

- Inflar la UI de la actividad con setContentView()
- Agregar fragmentos
- Obtener referencias de views con findViewById()
- Bindear datos a los view
- Iniciar consultas iniciales a fuentes de datos
- Crear instancias de los componentes de tu arquitectura

Luego veremos que su parámetro, de tipo Bundle, nos permite guardar valores simples asociados a la actividad para luego restaurarlos.

## Método onStart()

Este método es llamado después de onCreate(). En el podemos escribir instrucciones asociadas a la UI, ya que la actividad es visible al usuario. Sin embargo no es muy usado ya que son pocos los casos donde es trascendental (registros de componentes, librerías, etc.) su implementación.

## Método onRestart()

El método onRestart()**se ejecuta solo cuando la actividad se ha puesto en estado de Detenida**. Este es ejecutado antes de onStart() y después de onStop(). Su uso no es muy popular, pero es de utilidad cuando deseas diferenciar entre una recreación y un reinicio.

## Método onResume()

Se ejecuta antes de que la actividad se ponga *En Marcha* para interactuar con el usuario en primer plano. Normalmente en su interior se suele ejecutar animaciones, iniciar la **aplicación Camara**, cargar datos actualizados, etc.

## Método onPause()

Como vimos en el diagrama, onPause() es llamado cuando el sistema quita del primer plano a la actividad y así entrar en el estado *Pausada*. Del cuadro de la sección anterior se deduce que en este estado la acción es parcialmente visible, así que en esta callback podemos ejecutar instrucciones de actualización de UI si el caso lo requiere.

## Método onStop()

Es llamado antes de llegar al estado *Detenida* donde la actividad no será visible para el usuario. En consecuencia, aquí minimizaremos los recursos y acciones que requieran visibilidad. Ejemplos de ello sería pausar animaciones o usar una baja frecuencia de obtención de ubicaciones si se usa Google Play services location APIs.

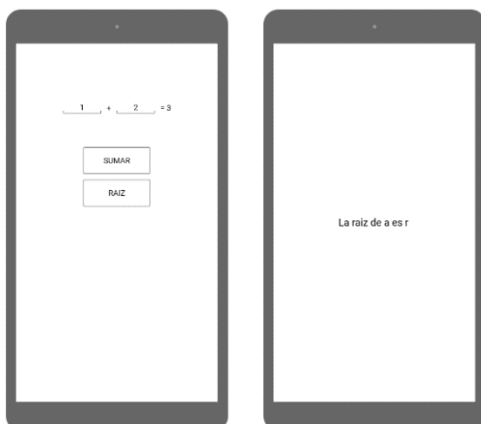
## Método onDestroy()

El sistema llama a este método antes de destruir la instancia de la actividad (estado *Inexistente*). En su interior incluiremos instrucciones de limpieza de recursos asociados (un buen ejemplo serían hilos generados en la creación).

**Importante:** Debes llamar al supermétodo de todos los métodos nombrados para que la actividad cumpla con sus acciones por defecto en el sistema, de lo contrario se lanzaran excepciones.

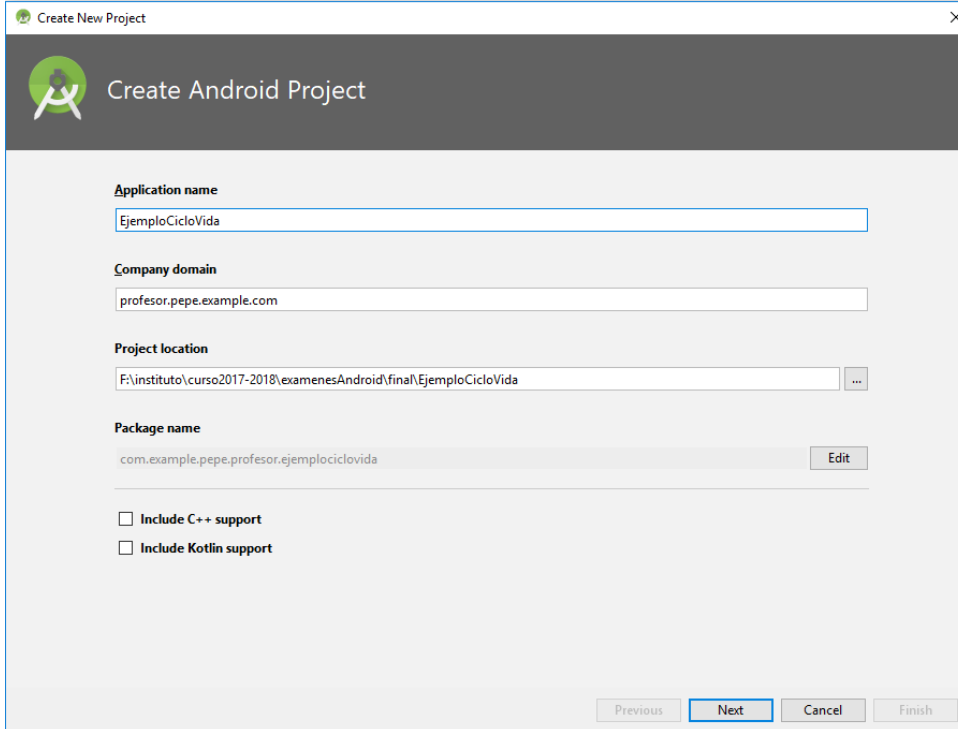
## Ejemplo De Actividades En Android Studio

Con el fin de que interiorices el conocimiento de las actividades en Android he creado un ejemplo en el cual usarás una actividad para sumar dos números al presionar un botón. Adicionalmente habrá otro botón para que obtengas la raíz del resultado de la suma, que se abrirá en otra activity distinta. La siguiente imagen ejemplifica rápidamente el propósito de la interacción de la UI:



## Crear Nuevo Proyecto Android Studio

Abre Android Studio y crea un nuevo proyecto llamado **Ejemplo Actividades**. Añade el paquete y ubicación que desees y presiona **Next**. En mi caso el setup me quedaría así:

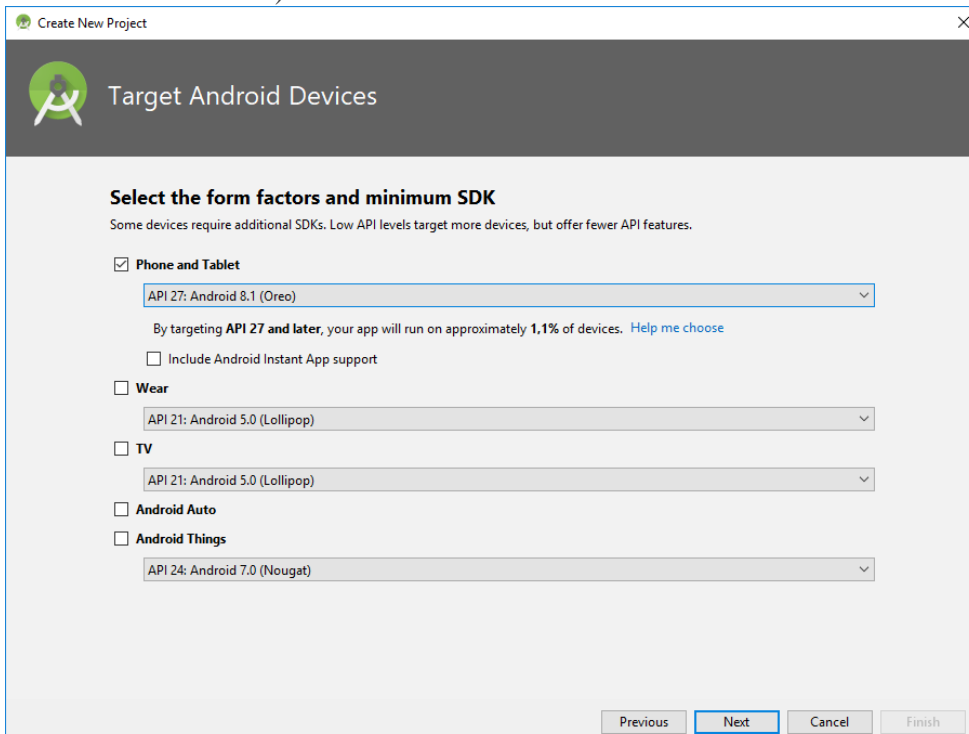


The screenshot shows the 'Create New Project' dialog in Android Studio. The title bar says 'Create New Project'. The main header area has the Android Studio logo and the text 'Create Android Project'. The form contains the following fields and options:

- Application name:** EjemploCicloVida
- Company domain:** profesor.pepe.example.com
- Project location:** F:\instituto\curso2017-2018\exámenesAndroid\final\EjemploCicloVida
- Package name:** com.example.pepe.profesor.ejemplociclovida (with an 'Edit' button)
- ☐ Include C++ support
- ☐ Include Kotlin support

At the bottom, there are four buttons: 'Previous', 'Next' (highlighted), 'Cancel', and 'Finish'.

En seguida, presiona **Next** dejando con el target por defecto que Android Studio nos sugiere (en el momento en que escribo el tutorial es la **API 15; Android 4.0.3 IceCreamSandwich**). Nosotros seleccionamos la 27:

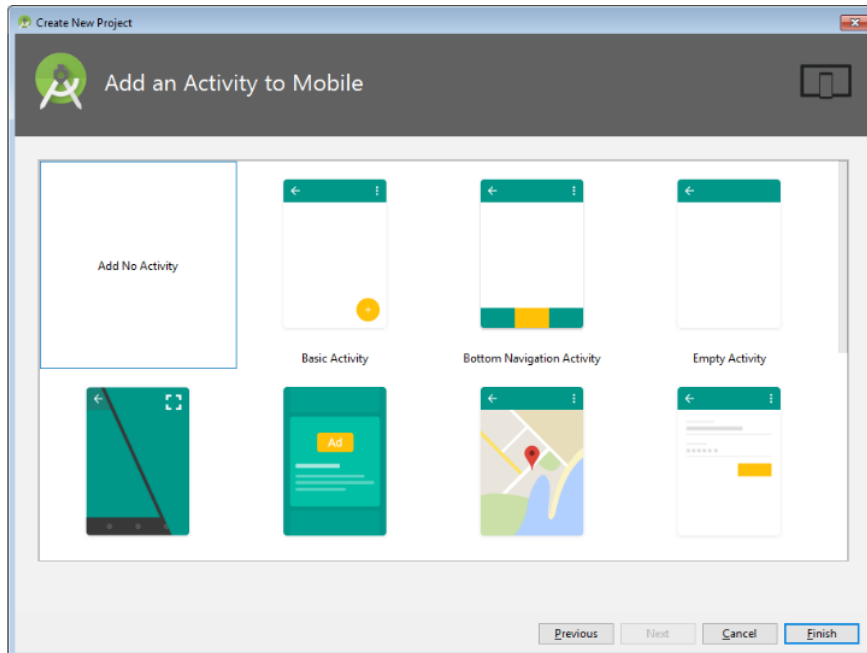


The screenshot shows the 'Target Android Devices' dialog in Android Studio. The title bar says 'Create New Project'. The main header area has the Android Studio logo and the text 'Target Android Devices'. The form contains the following options and dropdowns:

- Select the form factors and minimum SDK:** Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.
- ☒ **Phone and Tablet**: API 27: Android 8.1 (Oreo) (dropdown menu)
- ☐ **Wear**: API 21: Android 5.0 (Lollipop) (dropdown menu)
- ☐ **TV**: API 21: Android 5.0 (Lollipop) (dropdown menu)
- ☐ **Android Auto**
- ☐ **Android Things**: API 24: Android 7.0 (Nougat) (dropdown menu)

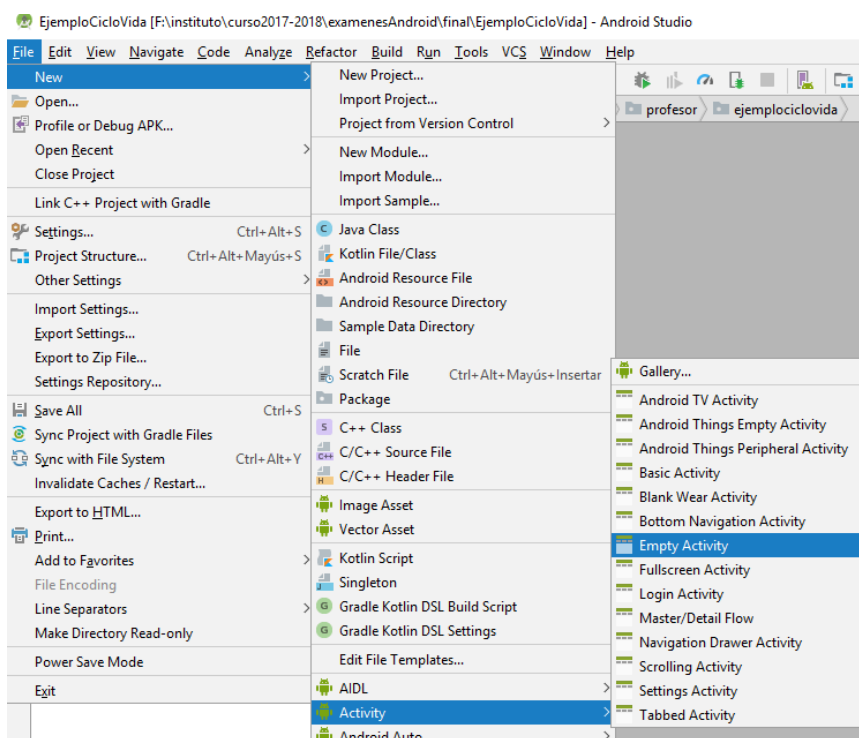
At the bottom, there are four buttons: 'Previous', 'Next' (highlighted), 'Cancel', and 'Finish'.

En la siguiente pantalla nos saldrá un menú con varios tipos de actividades prefabricados de Android Studio. Puedes elegir alguna de ellas para incorporarla como actividad principal inicial en tu nuevo proyecto. Sin embargo, nosotros marcaremos **Add No Activity** para comenzar en blanco y ver luego como añadir una actividad. Al finalizar presiona **Finish**.

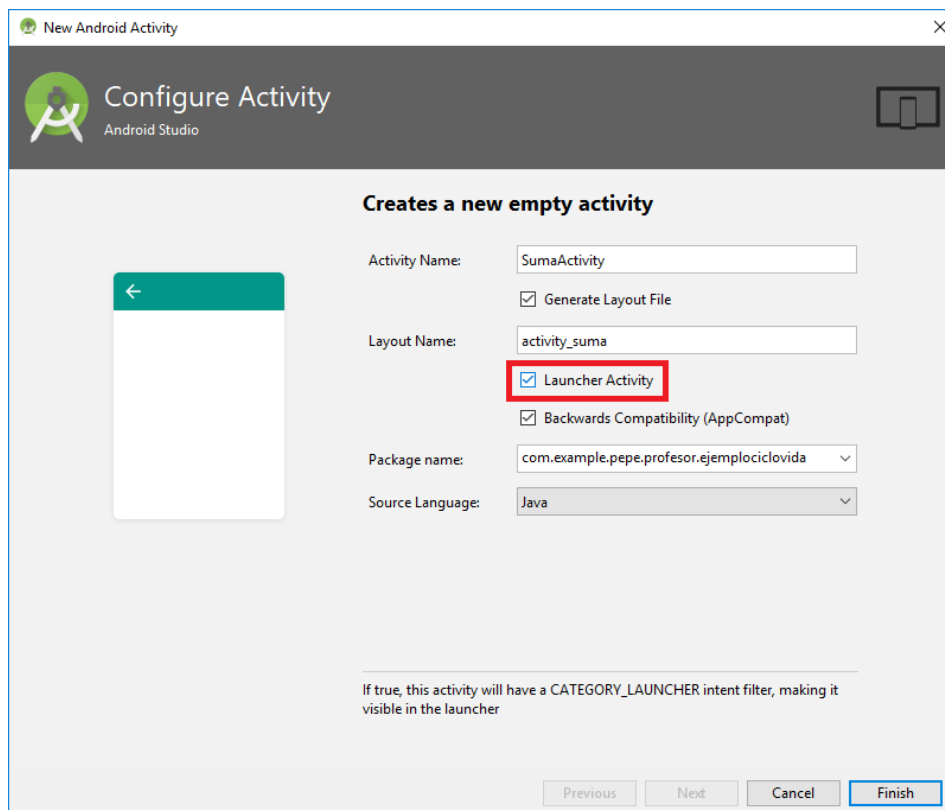


## Añadir Nueva Actividad

En el menú **File** seleccionamos **New > Activity > Empty Activity**. Este tipo de actividad está vacía, por lo que su layout se mantiene básico a diferencia de las demás opciones con otros fines especializados.

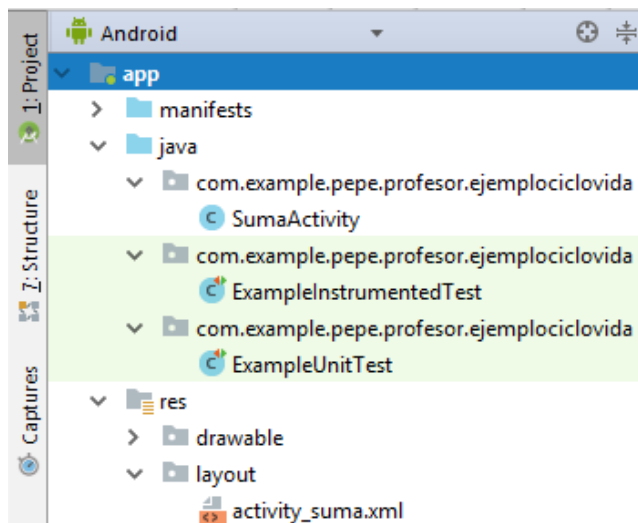


Ahora aparecerá un asistente que te guiará en la creación de la nueva actividad.



Esta será tu actividad para la suma de los dos números, así que llámala **SumaActivity**. Deja los demás datos por defecto y marca **Launcher Activity** para especificar en el manifiesto que será la principal. Confirma con **Finish**.

Si revisas el código de la actividad verás algo como esto:



```
public class SumaActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_suma);  
    }  
}
```

E incluso Android Studio añadió automáticamente la declaración en el AndroidManifest.xml.

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.herpro.ejemploactividades">  
  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic_launcher"  
        android:label="@string/app_name"  
        android:roundIcon="@mipmap/ic_launcher_round"  
        android:supportRtl="true"  
        android:theme="@style/AppTheme">  
        <activity android:name=".SumaActivity">  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>
```

## Modificar Layout Para La Suma

Abre el layout `res/activity_suma.xml` para agregar los views que viste en el boceto. Necesitaremos usar campos de edición (`EditText`) para recibir las entradas de los números, etiquetas para los símbolos (`TextView`) y el resultado; también botones (`Button`) para el cálculo de la suma y la apertura de la actividad de raíz. Todos ellos debes incluirlos en un `ConstraintLayout` para su distribución. Te comparto el código final por que todo esto lo aprenderemos más adelante:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".SumaActivity">

    <EditText
        android:id="@+id/number_a_input"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:hint="0"
        android:gravity="center"
        android:inputType="number"
        android:maxLength="2"
        app:layout_constraintEnd_toStartOf="@+id/plus_label"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="1" />

    <TextView
        android:id="@+id/plus_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:text="+"
        android:textAppearance="@style/TextAppearance.AppCompat.Large"
        app:layout_constraintBottom_toBottomOf="@+id/number_a_input"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.45"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.666" />

    <EditText
        android:id="@+id/number_b_input"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:hint="0"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:gravity="center"
        android:inputType="number"
        android:maxLength="2"
        app:layout_constraintEnd_toStartOf="@+id/sum_label"
        app:layout_constraintStart_toEndOf="@+id/plus_label"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="2" />

    <TextView
        android:id="@+id/sum_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginTop="8dp"
        android:textAppearance="@style/TextAppearance.AppCompat.Large"
        app:layout_constraintBottom_toBottomOf="@+id/number_b_input"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:text="= 0"
        tools:text="= 3" />

```



```

<Button
    android:id="@+id/add_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Sumar"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/number_b_input"
    app:layout_constraintVertical_bias="0.119" />

<Button
    android:id="@+id/get_root_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:text="Raíz"
    app:layout_constraintEnd_toEndOf="@+id/add_button"
    app:layout_constraintStart_toStartOf="@+id/add_button"
    app:layout_constraintTop_toBottomOf="@+id/add_button" />
</android.support.constraint.ConstraintLayout>

```

## Loguear Ciclo De Vida De SumaActivity

Para el posterior estudio de los diferentes estados que atraviesa la actividad SumaActivity debes loguear las callbacks vistas hasta ahora. Para ello sobrescribe cada método en el código Java y agrega el método Log.d() con un mensaje alusivo al método por el que transita.

```

public class SumaActivity extends AppCompatActivity {

    private static final String TAG = SumaActivity.class.getSimpleName();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_suma);
        Log.d(TAG, "onCreate()");
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "onStart()");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(TAG, "onRestart()");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.d(TAG, "onResume()");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d(TAG, "onPause()");
    }
}

```

```

@Override
protected void onStop() {
    super.onStop();
    Log.d(TAG, "onStop()");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy()");
}
}

```

## Obtener Instancias De Views Y Asignar Eventos

Como ves, aún no sumas los números que tu usuario estaría ingresando. Esto se debe a que no tienes las instancias de los campos de texto y tampoco has asignado una escucha al botón de suma. La solución entonces, consiste en tomar los views con `findViewById()` y setear una escucha anónima del `OnClickListener` al botón para operar los valores.

```

public class SumaActivity extends AppCompatActivity {

    private static final String TAG = SumaActivity.class.getSimpleName();
    private static final String STATE_SUM_OUTPUT = "sumOutput";

    private EditText mNumberAInput;
    private EditText mNumberBInput;
    private TextView mSumTotalLabel;
    private Button mAddButton;
    private Button mGetRootButton;

    private String sumOutput;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_suma);
        Log.d(TAG, "onCreate()");

        // Instancias
        mNumberAInput = findViewById(R.id.number_a_input);
        mNumberBInput = findViewById(R.id.number_b_input);
        mSumTotalLabel = findViewById(R.id.sum_label);
        mAddButton = findViewById(R.id.add_button);
        mGetRootButton = findViewById(R.id.get_root_button);

        // Actualización
        mAddButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                addUserNumbers();
            }
        });
    }

    private void addUserNumbers() {
        String inputA = mNumberAInput.getText().toString();
        String inputB = mNumberBInput.getText().toString();
        int numberA = inputA.isEmpty() ? 0 : Integer.parseInt(inputA);
        int numberB = inputB.isEmpty() ? 0 : Integer.parseInt(inputB);

        sumOutput = String.format(Locale.getDefault(), "%d", numberA + numberB);
        mSumTotalLabel.setText(sumOutput);
    }
}

```

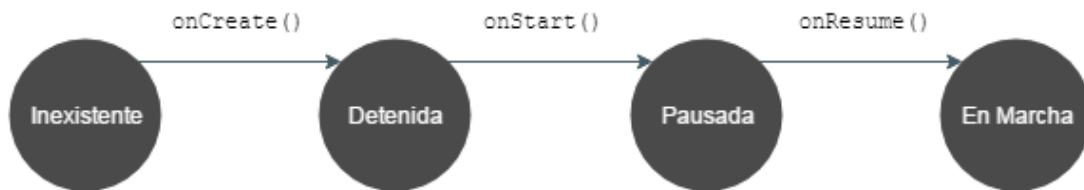
El valor de la suma junto al carácter '=' lo almacenaremos en una variable global llamada `sumOutput` para mantener su estado.

## Ejemplos De Ciclos De Vida

### Lanzamiento De Una Actividad

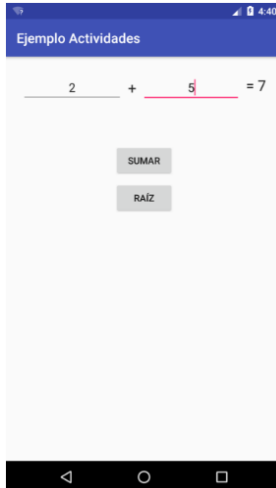
Este es el caso más básico, donde Android **abre un proceso nuevo** para nuestra app con el fin de referenciar las actividades. Como ya sabes, lanzar una app requiere que te sitúes en el **Launcher** y luego presiones el icono de la misma. Así que ejecuta la aplicación y verifica en qué orden se visualizan los métodos en el logcat de Android Studio después de la ejecución.

En mi logcat obtuve el recorrido clásico hacia la construcción de SumaActivity, pasando por los 4 estados de forma secuencial como ves en la imagen anterior. El siguiente diagrama me permite explicarme mejor:



### Navegación Hacia Atrás

Ahora después de tener la actividad en primer plano, inserta los 2 números y presiona el botón de suma. Si todo sale bien podrás ver el cálculo de la suma:



Seguidamente navega hacia atrás con el *Back Button* de la barra de navegación del sistema y fíjate en las callbacks del ciclo de vida que se visualizan. Como ves, ahora tienes la proyección reversa del caso de construcción: onPause() > onStop() > onDestroy():



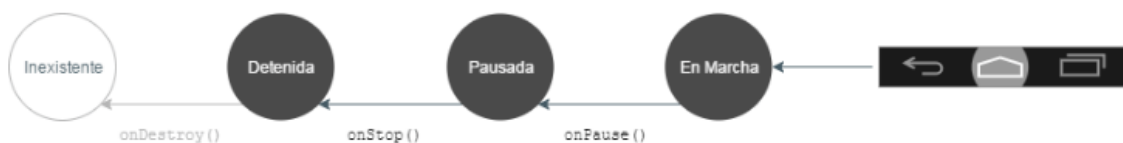
Vuelve a presionar el botón de la aplicación para ejecutarla:

¡Todos tus datos se han perdido!... lo cual, no debería ser una sorpresa. Cuando la actividad no estaba en primer plano llegó hasta su estado *Inexistente*, por lo que la instancia de la actividad fue destruida, lo que implica la destrucción de todo su contenido. Una vez reconstruida por el sistema, tu actividad generó una nueva instancia en memoria con sus valores por defecto.

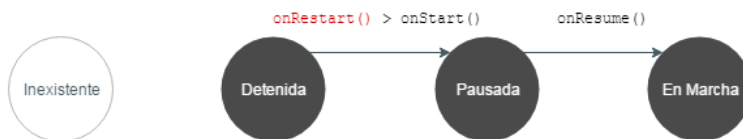
## Presionar El Home Button

Suma de nuevo dos números y esta vez prueba que estados recorre la actividad al presionar el botón Home de la barra de navegación.

Al terminar verás que solo se han llamado a la secuencia `onPause()` > `onStop()`. Esto se debe a que el propósito del Home button es ir al Launcher para cambiar de app y realizar otras acciones que el usuario desee.



El no llegar al estado Inexistente evita que la instancia de la actividad sea destruida, por lo que si abres de nuevo la aplicación presionando su icono verás que **se conservan los valores** de la suma. Esta vez se invoca `onRestart()` para diferenciar que la construcción viene de una instancia conservada.



## Presionar El Overview Button

Esta vez haz el mismo ejercicio pero **presionando el botón Overview** de la barra de navegación para abrir las [screen de recientes](#).

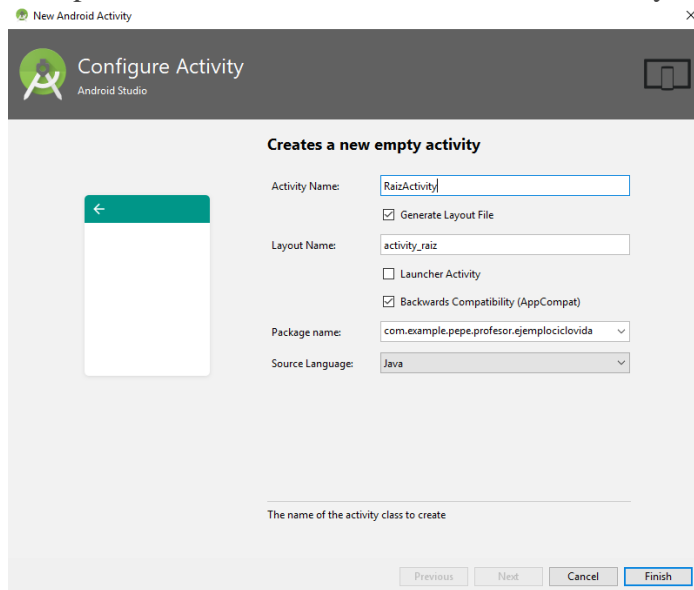
Notarás que el logcat evidencia que se invocan los mismos métodos que el anterior caso.



Al reiniciar tendremos exactamente el flujo `onRestart()` > `onStart()` > `onResume()` y la actividad mantendrá sus valores.

## Lanzar Otra Actividad En La Aplicación

Para probar este caso crea la actividad RaizActivity en el proyecto Android Studio.



Pon un TextView para asignar el resultado de la raíz en al archivo activity\_raiz.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".RaizActivity">
8
9      <TextView
10         android:id="@+id/zaiz_label"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:layout_marginEnd="8dp"
14         android:layout_marginStart="8dp"
15         android:layout_marginTop="16dp"
16         android:text=""
17         android:textAppearance="@style/TextAppearance.AppCompat.Large"
18         app:layout_constraintEnd_toEndOf="parent"
19         app:layout_constraintHorizontal_bias="0.101"
20         app:layout_constraintStart_toStartOf="parent"
21         app:layout_constraintTop_toTopOf="parent" />
22
23  </android.support.constraint.ConstraintLayout>

```

Por defecto se habrá creado la activity en el manifest.

Agregamos una escucha en la activity principal para el botón raíz:

```

mGetRootButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        showRootScreen();
    }
});

```

```

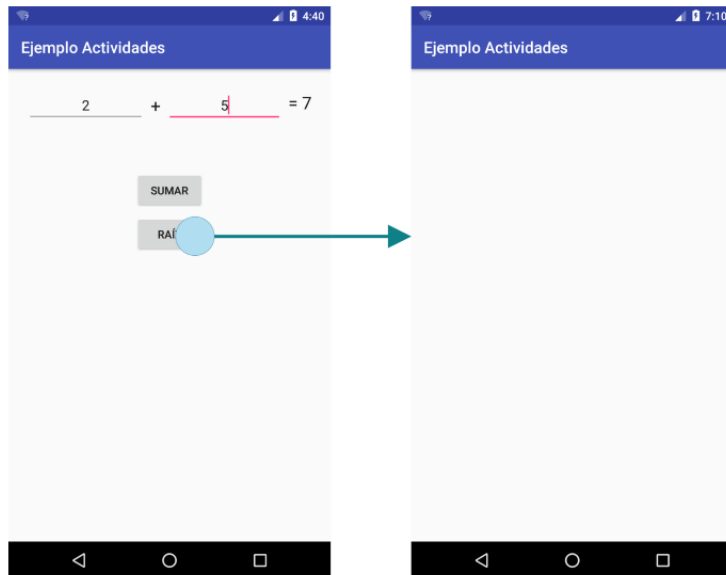
private void showRootScreen() {
    Intent intent = new Intent(this, RaizActivity.class);
    startActivity(intent);
}

```

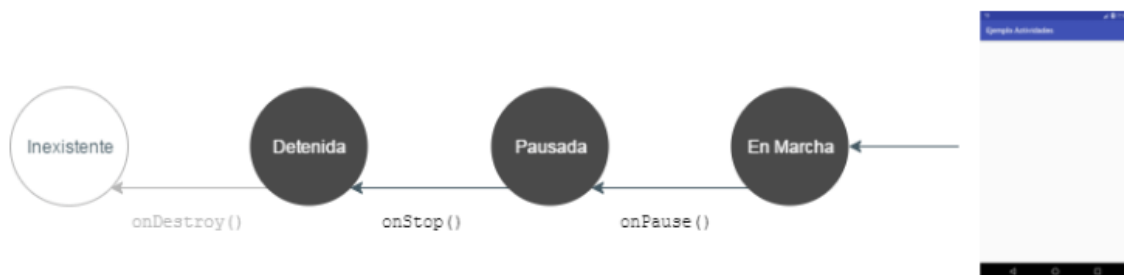
Atentos, no implementamos como calcular la raíz ni como visualizarla, solamente lanzamos otra activity en nuestra aplicación. Más adelante veremos como hacerlo, lo único que pretendemos con este ejemplo es comprender el ciclo de vida.

Ejecutamos la aplicación introduciendo los datos del ejemplo:

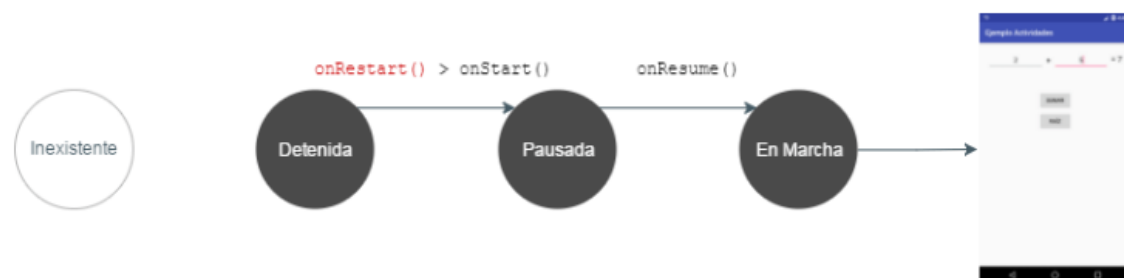
Y pulsamos el botón raíz, para que se abra la segunda activity:



El sistema conserva la instancia de la actividad de sumar como en los casos anteriores:



Y al pulsar el botón back:



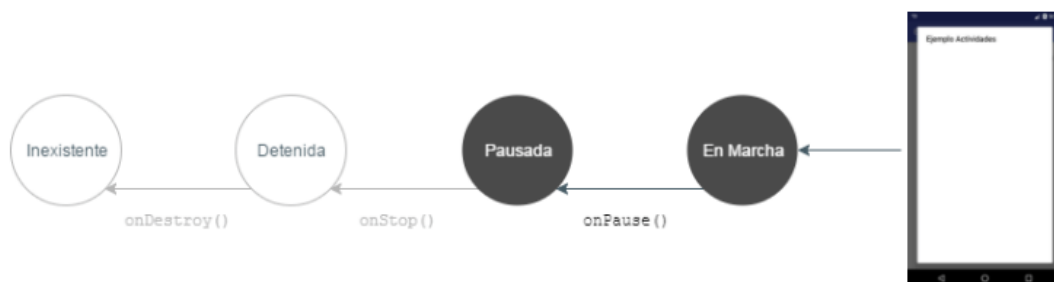
## Iniciar Una Actividad Con Estilo De Dialogo

El siguiente caso es cuando una actividad con estilo de diálogo aparece tomando el foco (no ocurre lo mismo con los diálogos comunes derivados de `DialogFragment`), pero nuestra actividad de suma aún es parcialmente visible.

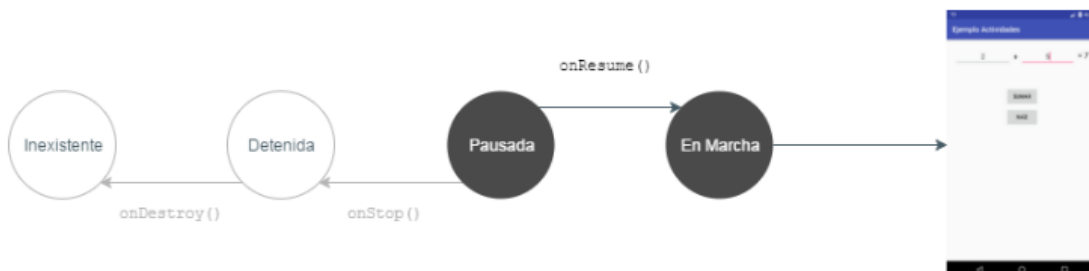
Para probarlo en `RaizActivity` ve al `AndroidManifest.xml` y agregar la siguiente línea en su nodo XML:

```
<activity
    android:name=".RaizActivity"
    android:theme="@style/Theme.AppCompat.Light.Dialog.Alert" />
```

Ejecuta la app y observa el ciclo de vida:

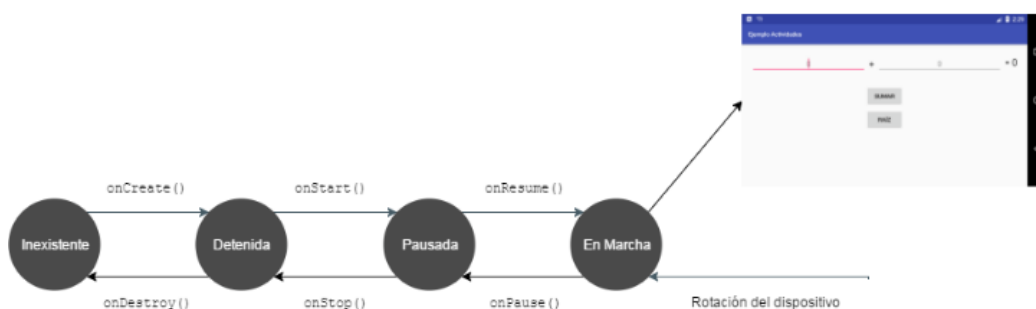


Tan solo se llamará a `onPause()` debido a que la actividad aún es visible, pero no está en primer plano. Por lo que si presionas Back será solo `onResume()` la callback faltante para llegar al estado *En Marcha*



## Cambios De Configuración

Un cambio de configuración fácil de percibir es la rotación de pantalla. Si rotas la actividad de suma verás que esta es reconstruida completamente, pasando por los 4 estados. Por lo que **perderemos el valor de nuestra variable de suma** (afortunadamente el framework nos ayuda a conservar el contenido de los `EditText`).



## Guardar Estado Simples De UI

Android te permite guardar el estado de las instancias mediante un objeto *Bundle*, el cual te permite insertar parejas clave-valor. El momento donde almacenamos dichos valores es en el método `onSaveInstanceState()` donde viene el parámetro `bundle`. Por ejemplo, para guardar el valor de la suma en nuestra actividad lo añadiremos con el método `Bundle.putString()` y le asignamos una clave asociada al valor:

```
public class SumaActivity extends AppCompatActivity {

    private static final String STATE_SUM_OUTPUT = "sumOutput";

    ...

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        Log.d(TAG, "onSaveInstanceState()");

        outState.putString(STATE_SUM_OUTPUT, sumOutput);
    }

    ...
}
```

## Restaurar Estado De La Actividad

Ya guardada nuestra variable antes de ser destruida la actividad, ahora solo queda restaurarla cuando se vuelva a crear la instancia. Esto lo podemos hacer en `onCreate()`, ya que su parámetro es el mismo `Bundle` que contiene la colección de estados tomados en `onSaveInstanceState()`. Para ello comprobamos que dicho objeto no sea `null` (puede que no sea una recreación) y usamos los métodos `Bundle.get*()` para obtener los valores:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_suma);
    Log.d(TAG, "onCreate()");

    // Instancias
    mNumberAInput = findViewById(R.id.number_a_input);
    mNumberBInput = findViewById(R.id.number_b_input);
    mSumTotalLabel = findViewById(R.id.sum_label);
    mAddButton = findViewById(R.id.add_button);
    mGetRootButton = findViewById(R.id.get_root_button);

    if(savedInstanceState!=null){
        sumOutput = savedInstanceState.getString(STATE_SUM_OUTPUT);
    }else {
        sumOutput = "= 0";
    }

    // Actualización
    mSumTotalLabel.setText(sumOutput);
    mAddButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            addUserNumbers();
        }
    });
    mGetRootButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            showRootScreen();
        }
    });
}
```

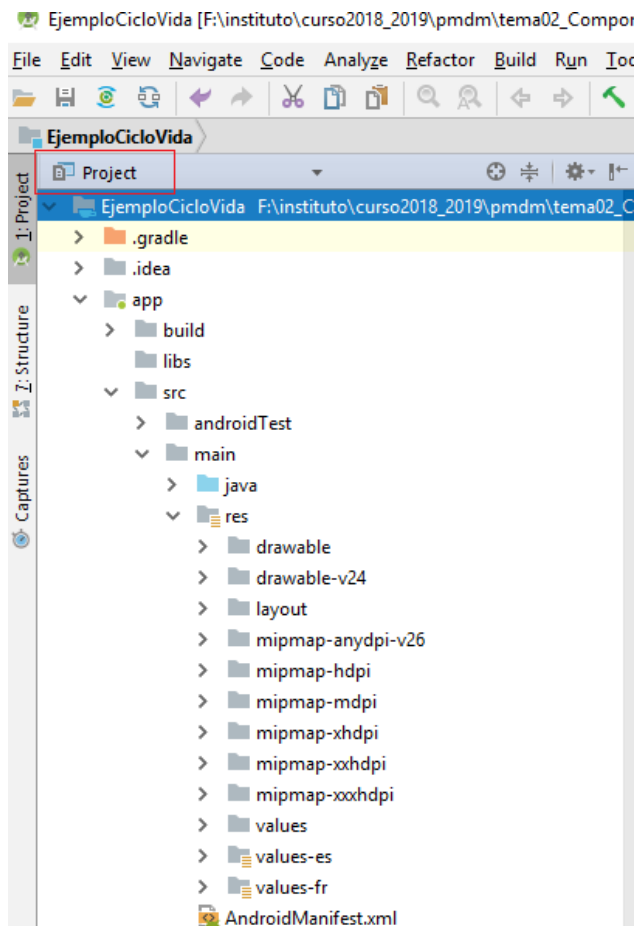


Otra opción para cargar el estado guardado es usar el método `onRestoreInstanceState()` el cual recibe el mismo Bundle solo si se detectó un guardado antes de la destrucción. Este método es ejecutado luego de `onStart()` y te evita comprobar nulidad. Normalmente usaremos `onCreate()` para restaurar los estados, pero habrá ocasiones especiales donde este método se ajustará mejor a tus necesidades si tu actividad posee alguna condición particular

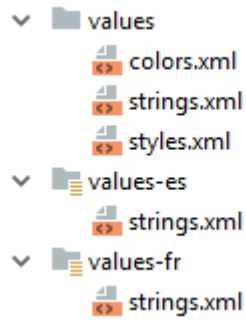
```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    sumOutput = savedInstanceState.getString(STATE_SUM_OUTPUT);
}
```

## Múltiples lenguajes

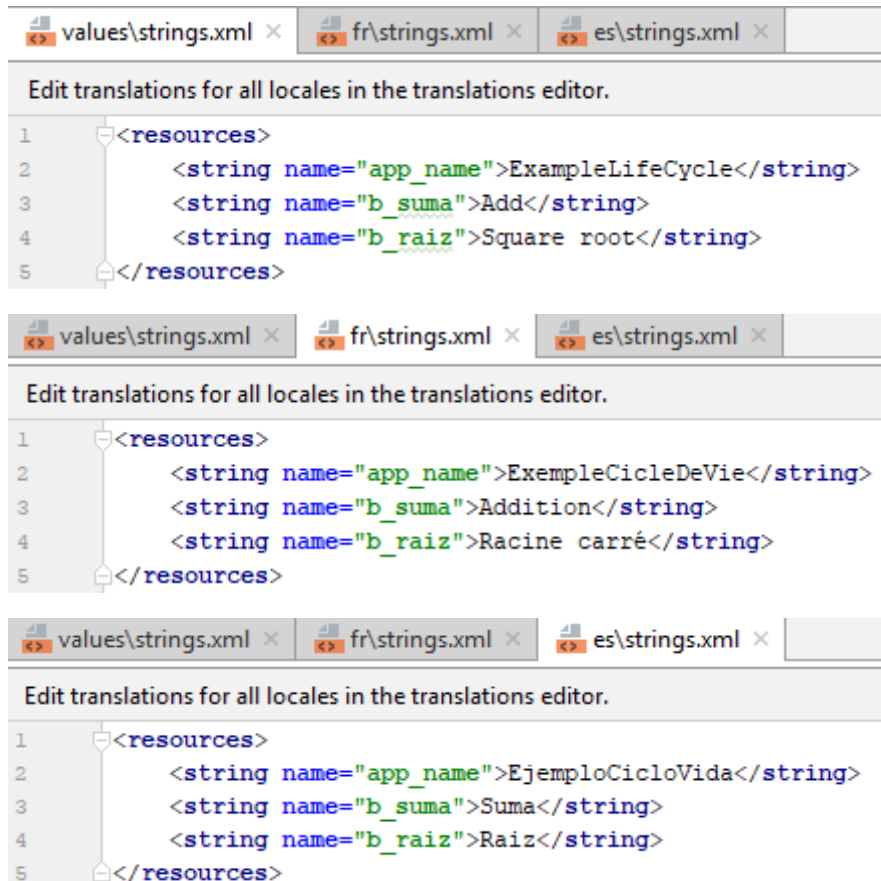
Vamos a configurar nuestra aplicación para poder trabajar en inglés, francés y español. Crearemos la siguiente estructura de archivos dentro de resources, para ello cambiamos a la vista de Project (en vez de Android):



Y generamos las carpetas `values-es` y `values-fr`, que contendrán respectivamente los strings de configuración de la aplicación y de los botones en sus respectivos idiomas.



Cada uno de estos archivos strings.xml contendrá los siguientes valores:



Probar los cambios cambiando el idioma en la configuración de vuestro terminal, automáticamente se producirán los cambios en vuestra aplicación